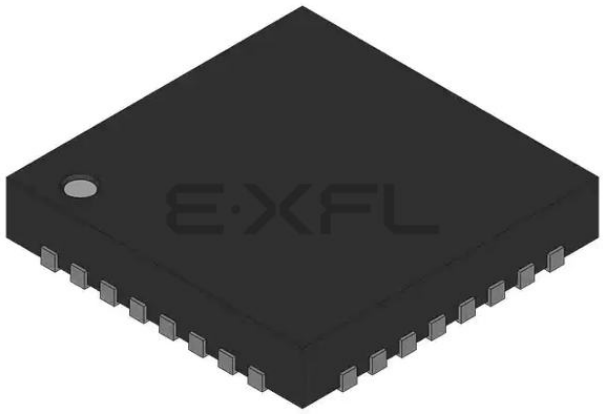


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	23
Program Memory Size	8KB (4K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	32-VFQFN Exposed Pad
Supplier Device Package	32-QFN (5x5)
Purchase URL	https://www.e-xfl.com/product-detail/atmel/atmega88-15mz

5.4 I/O Memory

The I/O space definition of the Atmel® ATmega48/88/168 is shown in Section “” on page 285.

All Atmel ATmega48/88/168 I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The Atmel ATmega48/88/168 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in opcode for the IN and OUT instructions. For the extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVR® the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

The I/O and peripherals control registers are explained in later sections.

5.4.1 General Purpose I/O Registers

The Atmel ATmega48/88/168 contains three general purpose I/O registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and status flags. General purpose I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

5.4.2 General Purpose I/O Register 2 – GPIOR2

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5.4.3 General Purpose I/O Register 1 – GPIOR1

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5.4.4 General Purpose I/O Register 0 – GPIOR0

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7.2 Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter idle mode, stopping the CPU but allowing the SPI, USART, analog comparator, ADC, 2-wire serial interface, Timer/Counters, watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the timer overflow and USART transmit complete interrupts. If wake-up from the analog comparator interrupt is not required, the analog comparator can be powered down by setting the ACD bit in the analog comparator control and status register – ACSR. This will reduce power consumption in idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.3 ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC noise reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the 2-wire serial interface address watch, Timer/Counter2, and the watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart from the ADC conversion complete interrupt, only an external reset, a watchdog system reset, a watchdog interrupt, a brown-out reset, a 2-wire serial interface address match, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT0 or INT1 or a pin change interrupt can wake up the MCU from ADC noise reduction mode.

7.4 Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter power-down mode. In this mode, the external oscillator is stopped, while the external interrupts, the 2-wire serial interface address watch, and the watchdog continue operating (if enabled). Only an external reset, a watchdog system reset, a watchdog interrupt, a brown-out reset, a 2-wire serial interface address match, an external level interrupt on INT0 or INT1, or a pin change interrupt can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from power-down mode, the changed level must be held for some time to wake up the MCU. Refer to Section 11. “External Interrupts” on page 73 for details.

When waking up from power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the reset time-out period, as described in Section 6.2 “Clock Sources” on page 24.

7.5 Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter power-save mode. This mode is identical to power-down, with one exception:

If Timer/Counter2 is enabled, it will keep running during sleep. The device can wake up from either timer overflow or output compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK2, and the global interrupt enable bit in SREG is set.

If Timer/Counter2 is not running, power-down mode is recommended instead of power-save mode.

The Timer/Counter2 can be clocked both synchronously and asynchronously in power-save mode. If Timer/Counter2 is not using the asynchronous clock, the Timer/Counter oscillator is stopped during sleep. If Timer/Counter2 is not using the synchronous clock, the clock source is stopped during sleep. Note that even if the synchronous clock is running in power-save, this clock is only available for Timer/Counter2.

The most typical and general program setup for the reset and interrupt vector addresses in Atmel® ATmega48 is:

```
Address      Labels Code                 Comments
0x000               rjmp   RESET        ; Reset Handler
0x001               rjmp   EXT_INT0     ; IRQ0 Handler
0x002               rjmp   EXT_INT1     ; IRQ1 Handler
0x003               rjmp   PCINT0       ; PCINT0 Handler
0x004               rjmp   PCINT1       ; PCINT1 Handler
0x005               rjmp   PCINT2       ; PCINT2 Handler
0x006               rjmp   WDT          ; Watchdog Timer Handler
0x007               rjmp   TIM2_COMPA   ; Timer2 Compare A Handler
0x008               rjmp   TIM2_COMPB   ; Timer2 Compare B Handler
0x009               rjmp   TIM2_OVF     ; Timer2 Overflow Handler
0x00A               rjmp   TIM1_CAPT    ; Timer1 Capture Handler
0x00B               rjmp   TIM1_COMPA   ; Timer1 Compare A Handler
0x00C               rjmp   TIM1_COMPB   ; Timer1 Compare B Handler
0x00D               rjmp   TIM1_OVF     ; Timer1 Overflow Handler
0x00E               rjmp   TIM0_COMPA   ; Timer0 Compare A Handler
0x00F               rjmp   TIM0_COMPB   ; Timer0 Compare B Handler
0x010               rjmp   TIM0_OVF     ; Timer0 Overflow Handler
0x011               rjmp   SPI_STC      ; SPI Transfer Complete Handler
0x012               rjmp   USART_RXC    ; USART, RX Complete Handler
0x013               rjmp   USART_UDRE   ; USART, UDR Empty Handler
0x014               rjmp   USART_TXC    ; USART, TX Complete Handler
0x015               rjmp   ADC          ; ADC Conversion Complete Handler
0x016               rjmp   EE_RDY       ; EEPROM Ready Handler
0x017               rjmp   ANA_COMP     ; Analog Comparator Handler
0x018               rjmp   TWI          ; 2-wire Serial Interface Handler
0x019               rjmp   SPM_RDY      ; Store Program Memory Ready Handler
;
0x01A  RESET:       ldi    r16, high(RAMEND); Main program start
0x01B               out    SPH,r16      ; Set Stack Pointer to top of RAM
0x01C               ldi    r16, low(RAMEND)
0x01D               out    SPL,r16
0x01E               sei                 ; Enable interrupts
0x01F               <instr>  xxx
  ...   ...    ...    ...
```

9.2 Interrupt Vectors in ATmega88

Table 9-2. Reset and Interrupt Vectors in ATmega88

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x001	INT0	External interrupt request 0
3	0x002	INT1	External interrupt request 1
4	0x003	PCINT0	Pin change interrupt request 0
5	0x004	PCINT1	Pin change interrupt request 1
6	0x005	PCINT2	Pin change interrupt request 2
7	0x006	WDT	Watchdog time-out interrupt

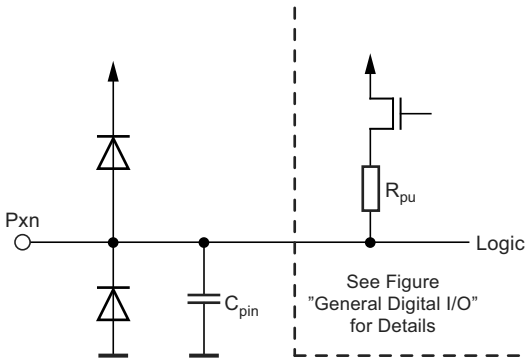
Notes:
1. When the BOOTRST fuse is programmed, the device will jump to the boot loader address at reset, see Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229.
2. When the IVSEL bit in MCUCR is set, interrupt vectors will be moved to the start of the boot flash section. The address of each interrupt vector will then be the address in this table added to the start address of the boot flash section.

10. I/O-Ports

10.1 Introduction

All AVR® ports have true read-modify-write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and ground as indicated in Figure 10-1. Refer to Section 26. “Electrical Characteristics” on page 260 for a complete list of parameters.

Figure 10-1. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in port B, here documented generally as PORTxn. The physical I/O registers and bit locations are listed in Section 10.4 “Register Description for I/O Ports” on page 71.

Three I/O memory address locations are allocated for each port, one each for the data register – PORTx, data direction register – DDRx, and the port input pins – PINx. The port input pins I/O location is read only, while the data register and the data direction register are read/write. However, writing a logic one to a bit in the PINx register, will result in a toggle in the corresponding bit in the data register. In addition, the pull-up disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

Using the I/O port as general digital I/O is described in Section 10.2 “Ports as General Digital I/O” on page 58. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in Section 10.3 “Alternate Port Functions” on page 62. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

- **Bit 1 - PCIF1: Pin Change Interrupt Flag 1**

When a logic change on any PCINT14..8 pin triggers an interrupt request, PCIF1 becomes set (one). If the I-bit in SREG and the PCIE1 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 0 - PCIF0: Pin Change Interrupt Flag 0**

When a logic change on any PCINT7..0 pin triggers an interrupt request, PCIF0 becomes set (one). If the I-bit in SREG and the PCIE0 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

11.6 Pin Change Mask Register 2 – PCMSK2

Bit	7	6	5	4	3	2	1	0	
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT23..16: Pin Change Enable Mask 23..16**

Each PCINT23..16-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

11.7 Pin Change Mask Register 1 – PCMSK1

Bit	7	6	5	4	3	2	1	0	
	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved Bit**

This bit is an unused bit in the Atmel® ATmega48/88/168, and will always read as zero.

- **Bit 6..0 – PCINT14..8: Pin Change Enable Mask 14..8**

Each PCINT14..8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT14..8 is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT14..8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

11.8 Pin Change Mask Register 0 – PCMSK0

Bit	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Figure 12-9 shows the same timing data, but with the prescaler enabled.

Figure 12-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

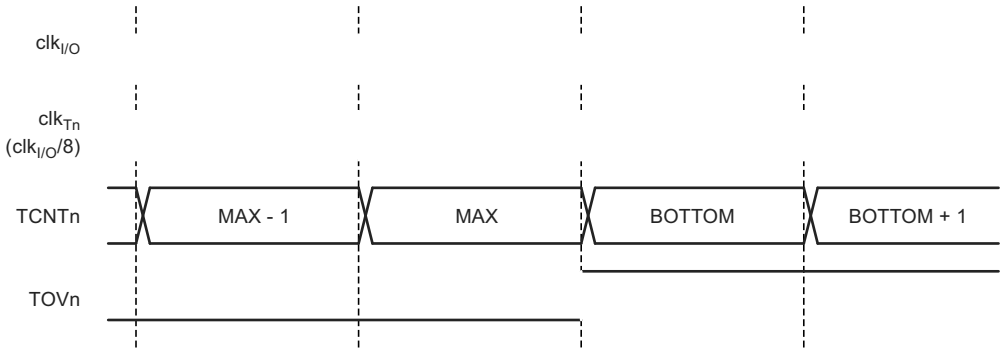


Figure 12-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 12-10.Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

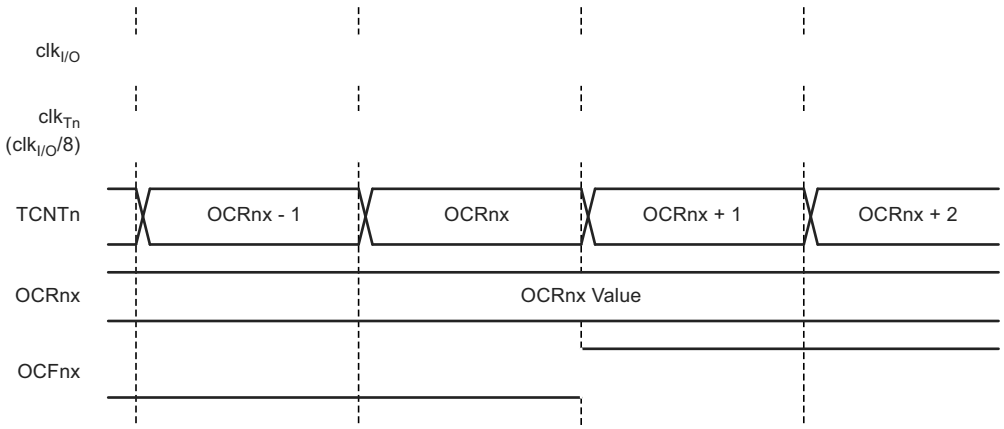
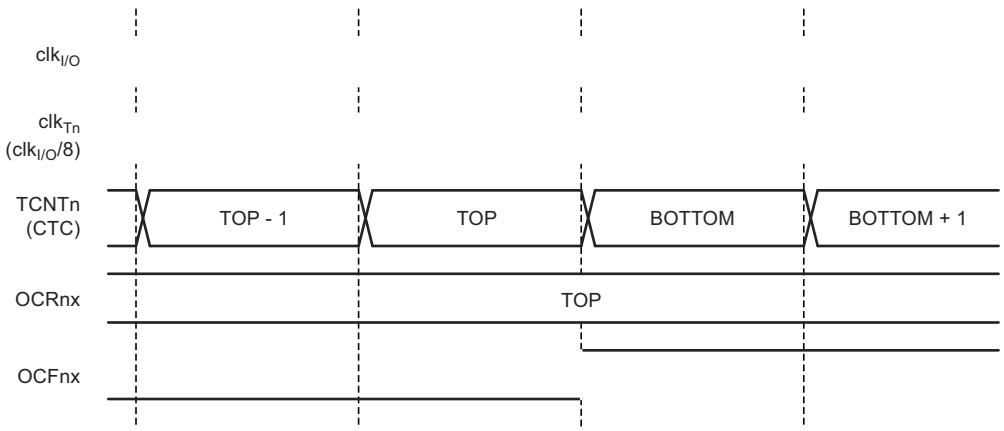


Figure 12-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 12-11.Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)

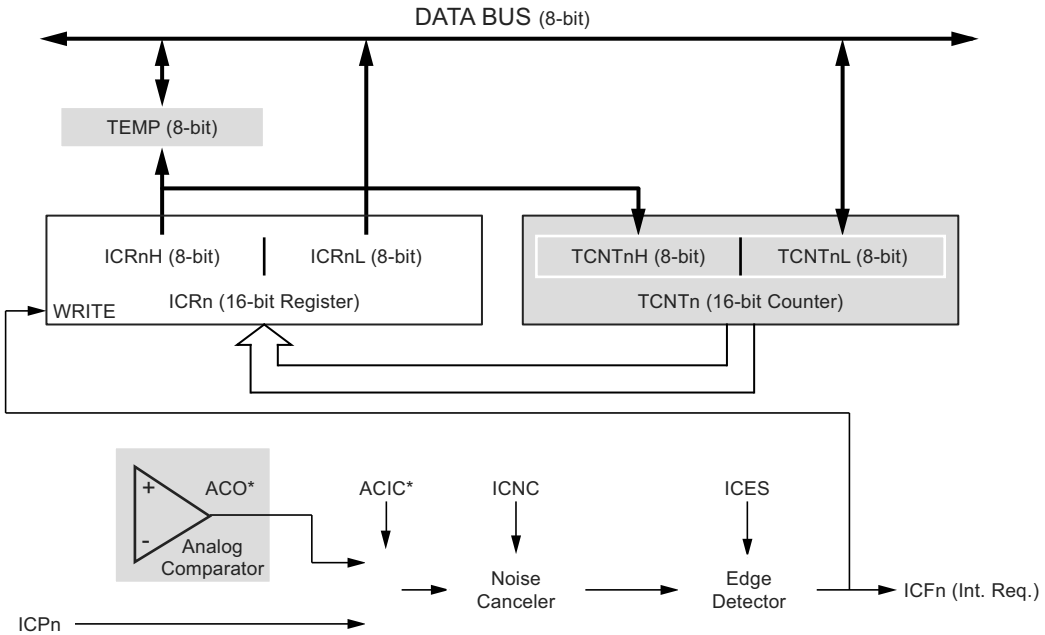


14.5 Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog-comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The input capture unit is illustrated by the block diagram shown in Figure 14-3. The elements of the block diagram that are not directly a part of the input capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

Figure 14-3. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the input capture pin (ICP1), alternatively on the analog comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the input capture register (ICR1). The input capture flag (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (ICIE1 = 1), the Input Capture flag generates an input capture interrupt. The ICF1 flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the input capture register (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP register.

The ICR1 register can only be written when using a waveform generation mode that utilizes the ICR1 register for defining the counter’s TOP value. In these cases the waveform generation mode (WGM13:0) bits must be set before the TOP value can be written to the ICR1 register. When writing the ICR1 register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

For more information on how to access the 16-bit registers refer to Section 14.2 “Accessing 16-bit Registers” on page 96.

14.10.5 Output Compare Register 1 A – OCR1AH and OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.10.6 Output Compare Register 1 B – OCR1BH and OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC1x pin.

The output compare registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 14.2 “Accessing 16-bit Registers” on page 96.

14.10.7 Input Capture Register 1 – ICR1H and ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The input capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the analog comparator output for Timer/Counter1). The input capture can be used for defining the counter TOP value.

The input capture register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 14.2 “Accessing 16-bit Registers” on page 96.

14.10.8 Timer/Counter1 Interrupt Mask Register – TIMSK1

Bit	7	6	5	4	3	2	1	0	
	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega48/88/168, and will always read as zero.

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 9. “Interrupts” on page 48) is executed when the ICF1 flag, located in TIFR1, is set.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are unused bits in the Atmel ATmega48/88/168, and will always read as zero.

The OCR2x register is double buffered when using any of the pulse width modulation (PWM) modes. For the normal and clear timer on compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR2x compare register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR2x register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2x buffer register, and if double buffering is disabled the CPU will access the OCR2x directly.

15.4.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOC2x) bit. Forcing compare match will not set the OCF2x flag or reload/clear the timer, but the OC2x pin will be updated as if a real compare match had occurred (the COM2x1:0 bits settings define whether the OC2x pin is set, cleared or toggled).

15.4.2 Compare Match Blocking by TCNT2 Write

All CPU write operations to the TCNT2 register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2x to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

15.4.3 Using the Output Compare Unit

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the output compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2x value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is downcounting.

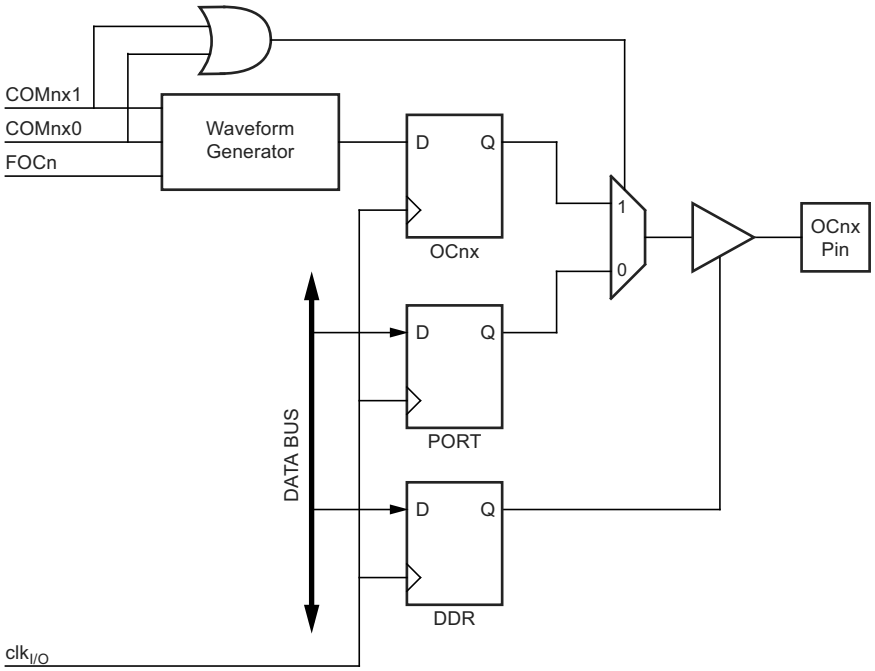
The setup of the OC2x should be performed before setting the data direction register for the port pin to output. The easiest way of setting the OC2x value is to use the force output compare (FOC2x) strobe bit in normal mode. The OC2x register keeps its value even when changing between waveform generation modes.

Be aware that the COM2x1:0 bits are not double buffered together with the compare value. Changing the COM2x1:0 bits will take effect immediately.

15.5 Compare Match Output Unit

The compare output mode (COM2x1:0) bits have two functions. The waveform generator uses the COM2x1:0 bits for defining the output compare (OC2x) state at the next compare match. Also, the COM2x1:0 bits control the OC2x pin output source. Figure 15-4 on page 123 shows a simplified schematic of the logic affected by the COM2x1:0 bit setting.
The I/O registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM2x1:0 bits are shown. When referring to the OC2x state, the reference is for the internal OC2x register, not the OC2x pin.

Figure 15-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the output compare (OC2x) from the waveform generator if either of the COM2x1:0 bits are set. However, the OC2x pin direction (input or output) is still controlled by the data direction register (DDR) for the port pin. The data direction register bit for the OC2x pin (DDR_OC2x) must be set as output before the OC2x value is visible on the pin. The port override function is independent of the waveform generation mode.

The design of the output compare pin logic allows initialization of the OC2x state before the output is enabled. Note that some COM2x1:0 bit settings are reserved for certain modes of operation. See Section 15.8 “8-bit Timer/Counter Register Description” on page 129

15.5.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM2x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2x1:0 = 0 tells the waveform generator that no action on the OC2x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 15-5 on page 130. For fast PWM mode, refer to Table 15-6 on page 130, and for phase correct PWM refer to Table 15-7 on page 130.

A change of the COM2x1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC2x strobe bits.

15.6 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the waveform generation mode (WGM22:0) and compare output mode (COM2x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM2x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2x1:0 bits control whether the output should be set, cleared, or toggled at a compare match (see Section 15.5 “Compare Match Output Unit” on page 122).

For detailed timing information refer to Section 15.7 “Timer/Counter Timing Diagrams” on page 127.

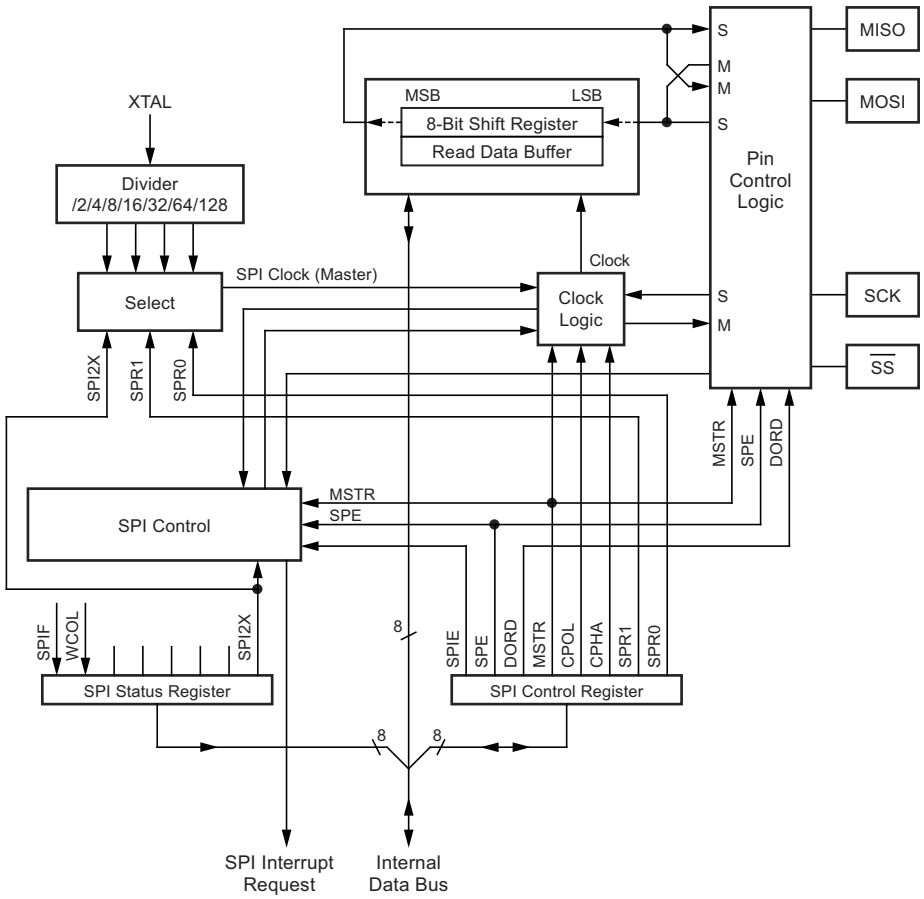
16. Serial Peripheral Interface – SPI

The serial peripheral interface (SPI) allows high-speed synchronous data transfer between the Atmel® ATmega48/88/168 and peripheral devices or between several AVR® devices. The Atmel ATmega48/88/168 SPI includes the following features:

- Full-duplex, three-wire synchronous data transfer
- Master or slave operation
- LSB first or MSB first data transfer
- Seven programmable bit rates
- End of transmission interrupt flag
- Write collision flag protection
- Wake-up from idle mode
- Double speed (CK/2) master SPI mode

The USART can also be used in master SPI mode, see Section 18. “USART in SPI Mode” on page 168. The PRSPI bit in Section 7.7.1 “Power Reduction Register - PRR” on page 35 must be written to zero to enable SPI module.

Figure 16-1. SPI Block Diagram(1)



Note: 1. Refer to Figure 1-1 on page 3, and Table 10-3 on page 64 for SPI pin placement.

Figure 16-3. SPI Transfer Format with CPHA = 0

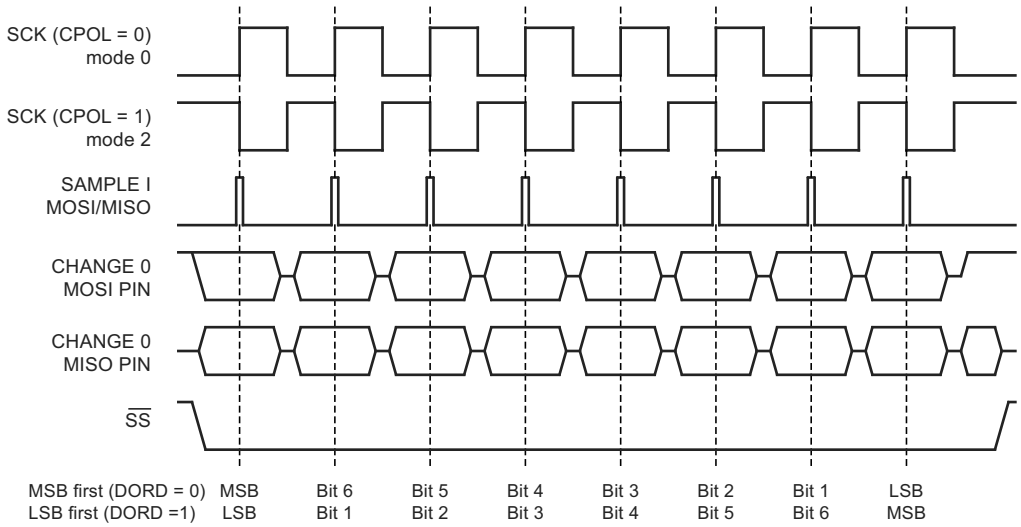
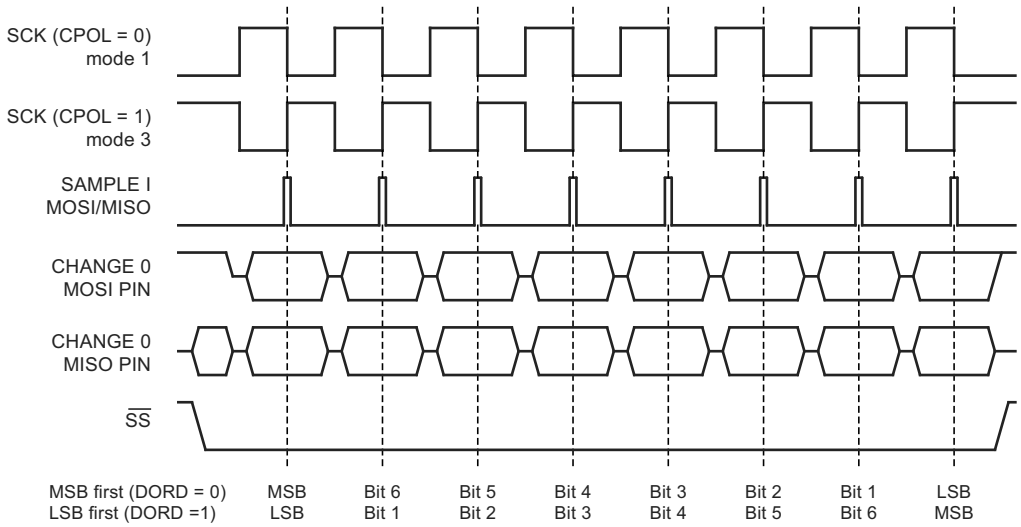


Figure 16-4. SPI Transfer Format with CPHA = 1



The function simply waits for data to be present in the receive buffer by checking the RXCn flag, before reading the buffer and returning the value.

17.6.2 Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZn=7) the ninth bit must be read from the RXB8n bit in UCSRnB before reading the low bits from the UDRn. This rule applies to the FEn, DORn and UPEn status flags as well. Read status from UCSRnA, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8n, FEn, DORn and UPEn bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both nine bit characters and the status bits.

Assembly Code Example[1]

```
USART_Receive:
        ; Wait for data to be received
        sbis   UCSRnA, RXCn
        rjmp   USART_Receive
        ; Get status and 9th bit, then data from buffer
        in     r18, UCSRnA
        in     r17, UCSRnB
        in     r16, UDRn
        ; If error, return -1
        andi   r18,(1<<FEn)|(1<<DORn)|(1<<UPEn)
        breq   USART_ReceiveNoError
        ldi    r17, HIGH(-1)
        ldi    r16, LOW(-1)
USART_ReceiveNoError:
        ; Filter the 9th bit, then return
        lsr    r17
        andi   r17, 0x01
        ret
```

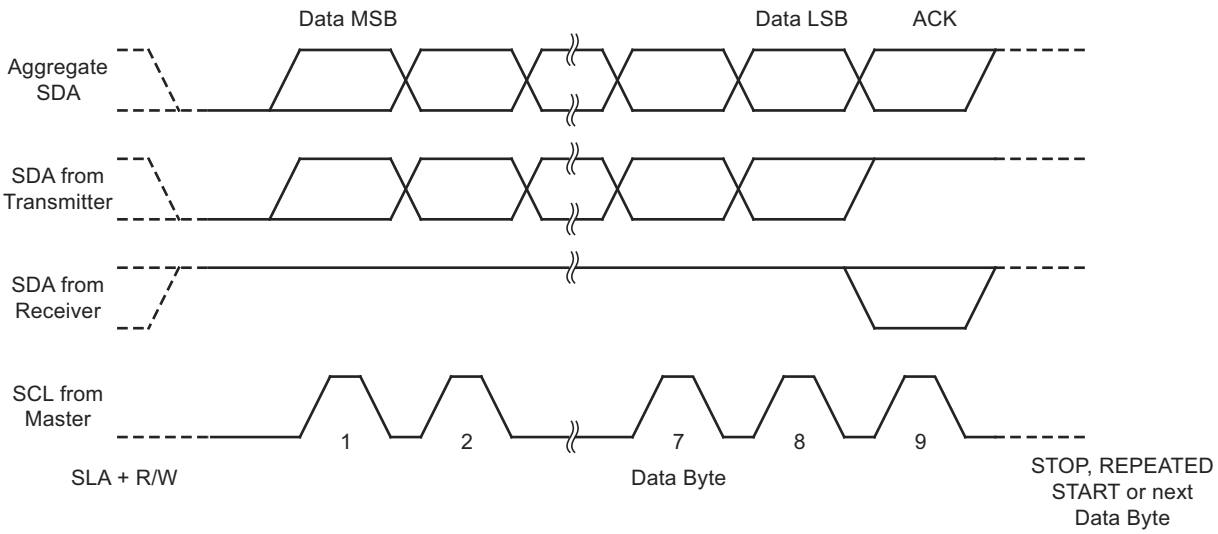
C Code Example[1]

```
unsigned int USART_Receive(void)
{
        unsigned char status, resh, resl;
        /* Wait for data to be received */
        while (!(UCSRnA & (1<<RXCn)))
        ;
        /* Get status and 9th bit, then data */
        /* from buffer */
        status = UCSRnA;
        resh = UCSRnB;
        resl = UDRn;
        /* If error, return -1 */
        if (status & (1<<FEn)|(1<<DORn)|(1<<UPEn))
                return -1;
        /* Filter the 9th bit, then return */
        resh = (resh >> 1) & 0x01;
        return ((resh << 8) | resl);
}
```

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The receive function example reads all the I/O registers into the register file before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

Figure 19-5. Data Packet Format

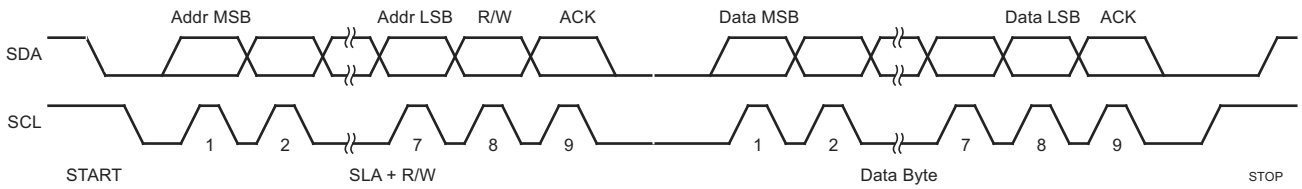


19.3.5 Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement handshaking between the master and the slave. The slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the master is too fast for the slave, or the slave needs extra time for processing between the data transmissions. The slave extending the SCL low period will not affect the SCL high period, which is determined by the master. As a consequence, the slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 19-6 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 19-6. Typical Data Transmission



19.4 Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e. the data being transferred on the bus must not be corrupted.
- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

22.4 Software Break Points

debugWIRE supports program memory break points by the AVR® break instruction. Setting a break point in AVR Studio® will insert a BREAK instruction in the program memory. The instruction replaced by the BREAK instruction will be stored. When program execution is continued, the stored instruction will be executed before continuing from the program memory. A break can be inserted manually by putting the BREAK instruction in the program.

The flash must be re-programmed each time a break point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of break points will therefore reduce the flash data retention. Devices used for debugging purposes should not be shipped to end customers.

22.5 Limitations of debugWIRE

The debugWIRE communication pin (dW) is physically located on the same pin as external reset (RESET). An external reset source is therefore not supported when the debugWIRE is enabled.

The debugWIRE system accurately emulates all I/O functions when running at full speed, i.e., when the program in the CPU is running. When the CPU is stopped, care must be taken while accessing some of the I/O registers via the debugger (AVR Studio).

A programmed DWEN fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN fuse should be disabled when debugWire is not used.

22.6 debugWIRE Related Register in I/O Memory

The following section describes the registers used with the debugWire.

22.6.1 debugWire Data Register – DWDR

Bit	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The DWDR register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

24. Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168

In Atmel® ATmega88 and Atmel ATmega168, the boot loader support provides a real read-while-write self-programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a flash-resident boot loader program. The boot loader program can use any available data interface and associated protocol to read code and write (program) that code into the flash memory, or read the code from the program memory. The program code within the boot loader section has the capability to write into the entire flash, including the boot loader memory. The boot loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the boot loader memory is configurable with fuses and the boot loader has two separate sets of boot lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

24.1 Boot Loader Features

- Read-while-write self-programming
- Flexible boot memory size
- High security (separate boot lock bits for a flexible protection)
- Separate fuse to select reset vector
- Optimized page(1) size
- Code efficient algorithm
- Efficient read-modify-write support

Note: 1. A page is a section in the flash consisting of several bytes (see Table 25-12 on page 247) used during programming. The page organization does not affect normal operation.

24.2 Application and Boot Loader Flash Sections

The flash memory is organized in two main sections, the application section and the boot loader section (see Figure 24-2 on page 231). The size of the different sections is configured by the BOOTSZ fuses as shown in Table 24-6 on page 240 and Figure 24-2 on page 231. These two sections can have different level of protection since they have different sets of lock bits.

24.2.1 Application Section

The application section is the section of the flash that is used for storing the application code. The protection level for the application section can be selected by the application boot lock bits (boot lock bits 0), see Table 24-2 on page 232. The application section can never store any boot loader code since the SPM instruction is disabled when executed from the application section.

24.2.2 BLS – Boot Loader Section

While the application section is used for storing the application code, the The boot loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire flash, including the BLS itself. The protection level for the boot loader section can be selected by the boot loader lock bits (boot lock bits 1), see Table 24-3 on page 232.

24.3 Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports read-while-write or if the CPU is halted during a boot loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ fuses as described above, the flash is also divided into two fixed sections, the read-while-write (RWW) section and the no read-while-write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 24-7 on page 240 and Figure 24-2 on page 231. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a boot loader software operation. The syntax “read-while-write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a boot loader software update.

24.3.1 RWW – Read-While-Write Section

If a boot loader software update is programming a page inside the RWW section, it is possible to read code from the flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the boot loader section. The boot loader section is always located in the NRWW section. The RWW section busy bit (RWWSB) in the store program memory control and status register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See Section 24.5.1 “Store Program Memory Control and Status Register – SPMCSR” on page 233 for details on how to clear RWWSB.

24.3.2 NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the boot loader software is updating a page in the RWW section. When the boot loader code updates the NRWW section, the CPU is halted during the entire page erase or page write operation.

Table 24-1. Read-While-Write Features

Which Section does the Z-pointer Address During the Programming?	Which Section Can be Read during Programming?	Is the CPU Halted?	Read-While-Write Supported?
RWW section	NRWW Section	No	Yes
NRWW section	None	Yes	No

Table 25-3. Lock Bit Protection Modes[1][2]. Only ATmega88/168.

BLB0 Mode	BLB02	BLB01	
1	1	1	No restrictions for SPM or LPM accessing the application section.
2	1	0	SPM is not allowed to write to the application section.
3	0	0	SPM is not allowed to write to the application section, and LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section.
4	0	1	LPM executing from the boot loader section is not allowed to read from the application section. If interrupt vectors are placed in the boot loader section, interrupts are disabled while executing from the application section.
BLB1 Mode	**BLB12**	**BLB11**	
1	1	1	No restrictions for SPM or LPM accessing the boot loader section.
2	1	0	SPM is not allowed to write to the boot loader section.
3	0	0	SPM is not allowed to write to the boot loader section, and LPM executing from the application section is not allowed to read from the boot loader section. If interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section.
4	0	1	LPM executing from the application section is not allowed to read from the boot loader section. If interrupt vectors are placed in the application section, interrupts are disabled while executing from the boot loader section.

Notes: 1. Program the fuse bits and boot lock bits before programming the LB1 and LB2.
2. “1” means unprogrammed, “0” means programmed

25.2 Fuse Bits

The Atmel® ATmega48/88/168 has three fuse bytes. Table 25-4 - Table 25-7 on page 244 describe briefly the functionality of all the fuses and how they are mapped into the fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 25-4. Extended Fuse Byte for ATmega48

Extended Fuse Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
–	1	–	1
SELFPRGEN	0	Self programming enable	1 (unprogrammed)

25.7.2 Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE fuse is programmed) and flash after a chip erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in flash or 256 byte EEPROM. This consideration also applies to signature bytes reading.

25.7.3 Chip Erase

The chip erase will erase the flash and EEPROM[1] memories plus lock bits. The lock bits are not reset until the program memory has been completely erased. The fuse bits are not changed. A chip erase must be performed before the flash and/or EEPROM are reprogrammed.

Note: 1. The EEPRPOM memory is preserved during chip erase if the EESAVE fuse is programmed.

Load command “chip erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for chip erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give $\overline{WR}$ a negative pulse. This starts the chip erase. RDY/$\overline{BSY}$ goes low.
6. Wait until RDY/$\overline{BSY}$ goes high before loading a new command.

25.7.4 Programming the Flash

The flash is organized in pages, see Table 25-12 on page 247. When programming the flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire flash memory:

A. Load command “write flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for write flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load address low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = address low byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load data low byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = data low byte (0x00 - 0xFF).
3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load data high byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = data high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch data

1. Set BS1 to “1”. This selects high data byte.
2. Give pAGEL a positive pulse. This latches the data bytes. (See Figure 25-3 on page 250 for signal waveforms)

26.2 DC Characteristics (Continued)

T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Condition	Symbol	Min.[5]	Typ.	Max.[5]	Unit
Power supply current[6]	Active 4MHz, V_{CC} = 3V (ATmega48/88/168L)	I_{CC}		1.8	3.0	mA
	Active 8MHz, V_{CC} = 5V (ATmega48/88/168)			6.0	10	mA
	Active 15MHz, V_{CC} = 5V (ATmega48/88/168)			10.0	16	mA
	Idle 4MHz, V_{CC} = 3V (ATmega48/88/168V)			0.4	1	mA
	Idle 8MHz, V_{CC} = 5V (ATmega48/88/168L)			1.4	2.4	mA
	Idle 15MHz, V_{CC} = 5V (ATmega48/88/168)			2.8	4	mA
Power-down mode	WDT enabled, V_{CC} = 3V			8	30	µA
	WDT enabled, V_{CC} = 5V			12.6	50	µA
	WDT disabled, V_{CC} = 3V			5	24	µA
	WDT disabled, V_{CC} = 5V			6.6	36	µA
Analog comparator input offset voltage	V_{CC} = 5V V_{in} = V_{CC}/2	V_{ACIO}		10	40	mV
Analog comparator input leakage current	V_{CC} = 5V V_{in} = V_{CC}/2	I_{ACLK}	–50		50	nA
Analog comparator propagation delay	V_{CC} = 4.5V	t_{ACID}		140		ns

Notes:
1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   Atmel ATmega48:
   1] The sum of all IOL, for ports C0 - C5, should not exceed 70mA.
   2] The sum of all IOL, for ports C6, D0 - D4, should not exceed 70mA.
   3] The sum of all IOL, for ports B0 - B7, D5 - D7, should not exceed 70mA.
   ATmega88/168:
   1] The sum of all IOL, for ports C0 - C5, should not exceed 100mA.
   2] The sum of all IOL, for ports C6, D0 - D4, should not exceed 100mA.
   3] The sum of all IOL, for ports B0 - B7, D5 - D7, should not exceed 100mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   ATmega48:
   1] The sum of all IOH, for ports C0 - C5, should not exceed 70mA.
   2] The sum of all IOH, for ports C6, D0 - D4, should not exceed 70mA.
   3] The sum of all IOH, for ports B0 - B7, D5 - D7, should not exceed 70mA.
   ATmega88/168:
   1] The sum of all IOH, for ports C0 - C5, should not exceed 100mA.
   2] The sum of all IOH, for ports C6, D0 - D4, should not exceed 100mA.
   3] The sum of all IOH, for ports B0 - B7, D5 - D7, should not exceed 100mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
5. All DC characteristics contained in this datasheet are based on actual ATmega88 microcontrollers characterization.
6. Values with Section 7.7.1 “Power Reduction Register - PRR” on page 35 enabled (0xEF).